



Apresentando, de forma prática, o framework React.js

O desenvolvimento web tem evoluído ao longo dos anos, através de novas tecnologias, novas ferramentas, novas formas de desenvolver, entre outros fatores, a fim de atender às diferentes necessidades por parte dos usuários, que devem ser sempre o foco de todo desenvolvimento. Se, por um lado, temos as empresas querendo atrair e manter seus clientes, vendendo seus produtos e oferecendo seus serviços, aqui também podemos pensar nos usuários internos de um sistema, chamados de clientes internos, capazes também, como os primeiros clientes citados, de fazer com que um projeto tenha sucesso ou não. Por outro lado, temos os desenvolvedores que devem estar atentos aos padrões, às boas práticas, aos novos recursos disponíveis, a fim de desenvolverem o melhor software com a maior qualidade possível e também com os menores custos. Logo, o processo de desenvolvimento deve envolver tanto os aspectos comerciais, pensando no lado da empresa, quanto aspectos de usabilidade, pensando nos clientes, internos e/ou externos, que utilizarão o software criado, assim como no lado de quem desenvolve os softwares.

Ao longo desse módulo, a partir da introdução acima, focaremos no desenvolvimento de software, focado na camada de front-end, de projetos web. Considerando tudo o que já foi dito até aqui, tenha em mente que, sempre que possível, devemos evitar “reinventar a roda”. Ou seja: fique atento às tendências de mercado no que diz respeito às boas práticas, ferramentas e outros recursos envolvidos no seu



processo de desenvolvimento. Falando apenas de ferramentas, uma iniciativa sempre recomendada e que costuma trazer muitos benefícios é a utilização de frameworks e/ou bibliotecas. Sua utilização nos economiza tempo, fornece soluções já testadas e aprovadas, além de nos permitir melhorar nossa forma de programar. Pensando nisso, veremos os fundamentos do framework React Js, uma biblioteca Javascript voltada para o desenvolvimento de interfaces ricas para aplicações Web. Aqui, cabe distinguir o que é um framework do que é uma biblioteca. Enquanto o framework é uma "solução completa", uma biblioteca se foca em resolver problemas específicos/fornecer soluções específicas. Logo, ao utilizar uma biblioteca, você provavelmente precisará utilizar outras. O React, como já dito, é uma biblioteca voltada para a criação de interfaces de usuário, com a qual criamos "Views" ricas e interativas de forma simples. Por outro lado, é possível adicionar outras bibliotecas, que trabalham em conjunto com o React, formando assim um ecossistema robusto para sua aplicação.

O React tem como uma de suas principais características ser baseado em componentes – escritos em Javascript. Na prática, isso significa que tanto os recursos de UI já fornecidos pela biblioteca, como os que desenvolvermos, devem ser voltados para componentes. Nesse sentido, podemos definir um componente como um pequeno pedaço de código que, seguindo boas práticas, deverá ser encapsulado e gerenciar seu próprio estado. Com isso, poderemos combinar inclusive reaproveitando, diferentes componentes em nossa aplicação, criando interfaces de usuário mais complexas.

Em complemento com o que foi exposto até aqui, podemos dizer que há, basicamente, dois pré-requisitos para trabalharmos com React.

1. Sabermos utilizar, em Javascript, recursos como variáveis, arrow functions, objetos, promises, entre outros;

2. Compreendermos quatro conceitos básicos do React:

- Components;
- JSX;
- State;
- Props.

Como nesse módulo não cabe tratarmos de Javascript, veremos então um pouco mais sobre os 4 principais conceitos do React. Antes de continuarmos, vale frisar que há duas formas de utilizarmos a biblioteca React: importando o ".js" da biblioteca para uma aplicação já existente (ou nova) ou instalarmos a biblioteca através de gerenciadores de pacote, como o npm. Ao longo deste módulo vamos considerar a segunda opção, cujo how-to pode ser encontrado na página oficial do projeto.

A partir daqui, trataremos dos fundamentos do React. Para começar, veja o fragmento de código abaixo:

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Primeiro componente React.
        </p>
        <p>
          Saiba mais sobre a biblioteca acessando sua documentação
                        oficial, no link abaixo.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Aprenda React
        </a>
      </header>
    </div>
  );
}

export default App;
```



Código-fonte: App.js: Componente React

Nesse script podemos ver implementado um componente React. Sobre o mesmo, aqui vão algumas observações:

- No início do script (.js), temos os imports. Aqui, como em linguagens como Java, importamos as bibliotecas e outros componentes que utilizaremos no script em questão;

- A seguir, temos uma function JS chamada App. Tal function poderia também ser escrita no formato de arrow function. Dentro da function App temos uma mistura de códigos Javascript e HTML (uma linguagem especial que chamamos de JSX);

- Ao final do script temos a instrução "export default App". A partir dela exportamos nossa function que poderá ser importada e utilizada em outras partes de nossa aplicação.

O código apresentado nos permite ter o primeiro contato com um Componente em React, assim como vemos, em primeira mão, como a linguagem JSX é formada (através da combinação de código Javascript e código HTML). Tenha em mente que, embora vamos escrever nossos componentes utilizando JSX, no momento de visualizarmos nossa aplicação no navegador, o que será visto será HTML e Javascript puros. Isso é possível graças ao processo de transpilação, realizado ao fazermos o build de nosso código. Nesse processo, todo o código JSX é transformado em Javascript e HTML. Em relação aos conceitos de State e Props, um componente React pode ter seus próprios estados, representados por variáveis e que podem ser acessados por outros componentes, caso o componente ao qual pertence o exponha para o resto da aplicação. Também podemos dizer que States são um conjunto de dados gerenciados (com o Hook useState) pelo próprio componente. Além disso, um componente pode também receber propriedades de outros componentes. Essas propriedades, na prática também representadas por variáveis, assim como por métodos, recebem o nome de Props. Veja, a seguir, novos exemplos de código, em que nosso primeiro componente foi fragmentado em dois componentes.



```
//O componente foi criado através de uma arrow function
//  Repare que o componente recebe duas props: titulo e descricao
const Texto = ({titulo, descricao}) => {
   return (
     <div>
           {/*Abaixo, dentro de {}, as duas Props recebidas do componente App.js são utilizadas*/}
       <h1>{titulo}</h1>
       <p>
         {descricao}
       </p>
     </div>
   );
}

export default Texto;
```

Código-fonte: Componente Texto.js

```
import logo from './logo.svg';
import './App.css';
import Texto from './Texto'; //aqui o novo componente criado, chamado Texto, é importado
import { useState } from 'react';

function App() {
  const [title, setTitle] = useState('Titulo Inicial do State');
  const [description, setDescription] = useState('Descricao Inicial do State');
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Primeiro componente React.
        </p>
        <p>
          Saiba mais sobre a biblioteca acessando sua documentação
                        oficial, no link abaixo.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Aprenda React
        </a>
      </header>
      <section>
            {/*Abaixo o componente Texto é chamado/utilizado*/}
            {/*Através das Props titulo e descricao, são passados para o componente os States title e
description*/}
        <Texto titulo={title} descricao={description} />
      </section>
    </div>
  );
}

export default App;
```

Código-fonte: Componente App.js



Perceba em cada código os comentários que foram inseridos, explicando o que acontece em algumas linhas em especial. Reproduzo, a seguir, tais comentários que apresentam a aplicação prática dos conceitos de Props e State, mencionados anteriormente.

- O componente Texto.js foi criado através de uma arrow function. Logo, veja que podemos criar um componente tanto declarando uma function, quanto usando uma arrow function;

- O componente Texto.js recebe dois parâmetros – que chamamos de Props. Tais propriedades são passadas para esse componente no momento em que foi inserido no componente App.js;

- No componente App.js importamos o componente Texto.js, na área em que as importações são realizadas;

- Foram criados, no componente App.js, dois States: title e description. Repare que os mesmos foram declarados como constantes. Além disso, ambos são React Hooks (veremos sobre Hooks mais adiante) useState;

- Os States declarados no App são externalizados para o componente Texto, onde serão tratados como

Props.



Após vermos os conceitos fundamentais do React, chegou a hora de entendermos como aplicar CSS nos componentes. Quando criamos um primeiro projeto React, o mesmo já contará com um componente inicial, o App.js e seu arquivo de estilos, o App.css. Repare, no componente App.js declarado acima, que o App.css está sendo importado. A seguir tal código é demonstrado. Repare que trata-se de código CSS semelhante ao que usamos juntamente com HTML. No mesmo são declaradas algumas classes, com suas respectivas propriedades. Em relação à utilização dessas classes nos componentes, usamos o atributo "className", passando como valor o nome da classe. Veja esse fragmento:

```
import logo from './logo.svg';
import './App.css';
  <!-- ... -->
  <div className="App">
    <header className="App-header">
  <!-- ... -->
```

Código-fonte: Utilizando className

No código acima, contido no componente App.js, podemos ver a definição de duas className: App e App-header, nas tags div e header, respectivamente.

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
```



Código-fonte: Arquivo CSS externo - App.css

Essa forma de estilizar os componentes React com CSS, através de className, é a mais recomendada, porém não é a única. Além de bibliotecas de terceiros, há ainda outra forma nativa de aplicar estilos: o CSS-in-JS. Para entendê-lo, vamos ver um novo fragmento de código:

```
//Estilo CSS sendo definido como objeto Javascript
const h1Style = {
        color: 'blue',
        fontSize: '36px',
};
const Texto = ({titulo, descricao}) => {
  return (
    <div>
        {/*Abaixo, a partir do atributo style, o objeto JS, contendo estilo CSS, é setado*/}
      <h1 style={h1Style}>{titulo}</h1>
      <p>
        {descricao}
      </p>
    </div>
  );
}

export default Texto;
```

Código-fonte: CSS-in-JS



No exemplo acima temos novamente o componente Texto.js, onde uma nova constante, chamada h1Style, foi declarada. Trata-se de um objeto Javascript. No mesmo, duas propriedades foram inseridas: color e fontSize. Repare, a partir desta segunda propriedade, que ela não segue o formato CSS padrão. Aqui o padrão camelCase precisa ser seguido. Essa constante é utilizada com o atributo style, na tag h1. Por fim, cabe reforçar que devemos dar preferência à declaração de estilos com className, deixando o CSS-in-JS para situações onde precisamos aplicar estilos de forma dinâmica, em tempo de renderização.

Chegamos ao final de mais um módulo, em que os conceitos iniciais da biblioteca React foram apresentados, além dos fundamentos da biblioteca conhecemos também sua aplicação prática.

Atividade Extra

Há muito mais para conhecermos sobre a biblioteca React. Nesse sentido, sugiro que você comece colocando a mão na massa, criando sua primeira aplicação em React. Para isso, veja os tutoriais disponíveis na documentação oficial, em português, disponíveis em:

Link: https://pt-br.reactjs.org/ **(Acesso em 28/09/2022)**

Referência Bibliográfica



MORGAN, Joe. **How to code in React.js.** New York City, NY: Digital Ocean, 2021.

Atividade Prática Módulo 8

Título da Prática: Customizando nosso primeiro componente React.

Objetivos: Possibilitar ao aluno entender como modificar um componente React.

Materiais, Métodos e Ferramentas: IDE VS Code

Atividade Prática

Quando criamos uma aplicação React, por padrão, são criadas algumas pastas e arquivos. Entre as pastas, a “src” é responsável por armazenar o código de nossos componentes. Normalmente, criamos outra pasta dentro dela, chamada pages. Sobre isso, veremos mais adiante.

Estando na pasta “src”, perceba que há um script chamado App.js. Trata-se de um componente que nos fornece um exemplo inicial de algumas características do React Js, como a linguagem JSX. Nessa atividade, você deverá modificar o componente em questão, removendo o conteúdo existente e inserindo novos conteúdos. A seguir, você deverá executar a aplicação e visualizar, no navegador,

as alterações realizadas. As etapas acima são descritas em detalhes a seguir:



1. Abra o projeto na IDE VS Code;

2. A partir do VS Code, com o projeto já aberto no mesmo, encontre e abra o App.js, que está dentro da pasta “src”;

3. Localize a instrução “return();” e remova todo o seu conteúdo, mantendo apenas o seu esqueleto – “return();”;

4. Dentro da instrução return – mais precisamente, dentro dos parênteses da mesma – insira o seguinte:

```
<div>

<h1>Primeira aplicação React</h1>

<p>Esse é um exemplo de um componente React.</p>

</div>
```

5. Salve as alterações;

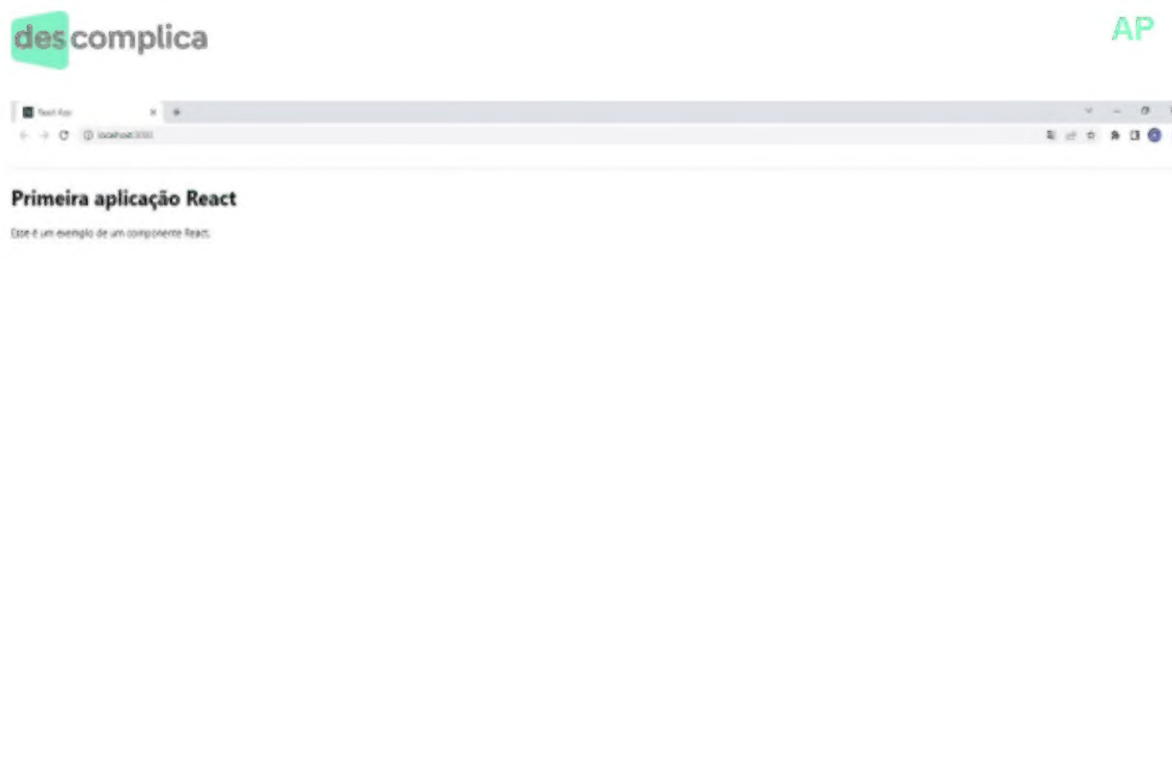
6. Através da linha de comando, estando na pasta do projeto, execute o comando “npm start”;

7. Visualize no navegador as alterações realizadas no componente App.js;

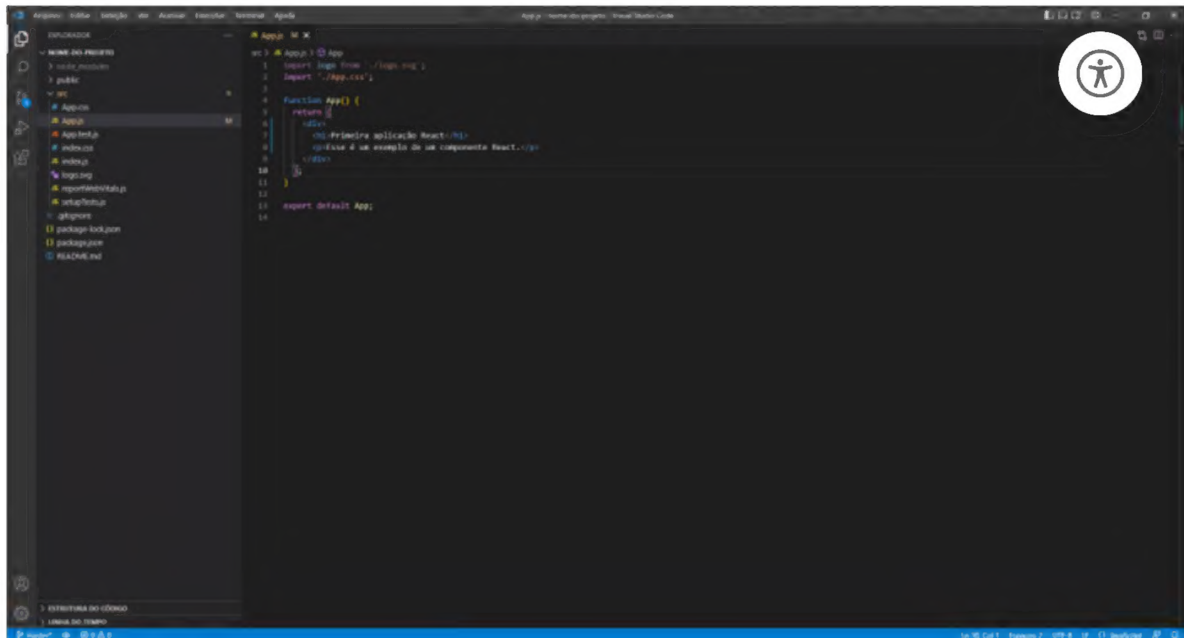


Gabarito Atividade Prática

Ao final da execução dos passos descritos acima, ao testar as alterações realizadas, o aluno deverá ver no navegador web, a seguinte página:



Na IDE VS Code, o código deverá estar desta forma:



Ir para exercício